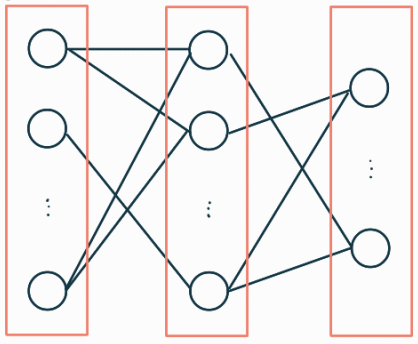
Many of neurons are connected to one another. We can create a complex network this way. We can have different types of structures.

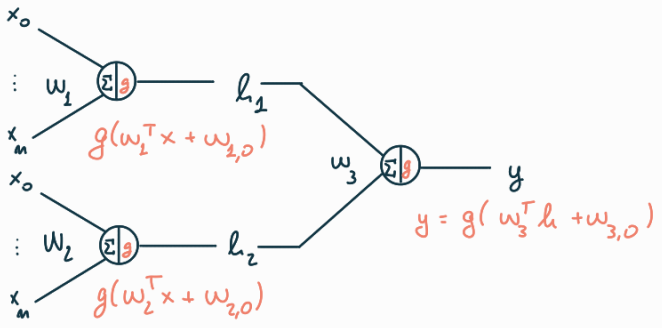
Feedforward Neural Networks are networks in which information flows in only one direction; we don't have ① loops (connections to previous layers) and ② jumps from one layer to another that is not the next one.



layers

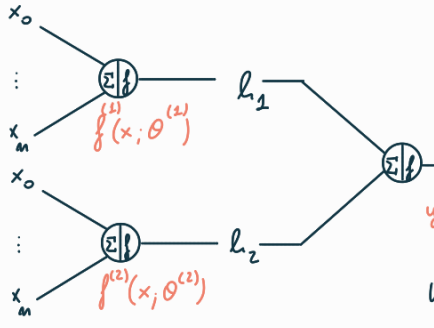
Let's start from the perceptron: a neuron for which the activation function is the sign function. This component can be written as $g(w^T x + w_0)$

Consider more neurons



This is a simple two-layer neural network
First layer → 2 units
Second layer → 1 unit

In more general terms:



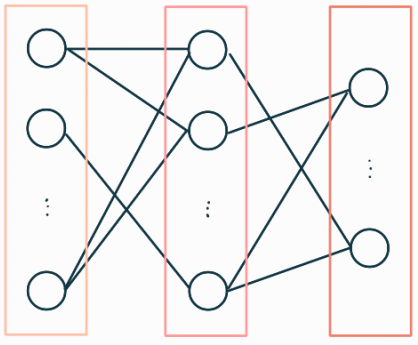
generalizing: h_1 will be the output of a parametric function $f^{(1)}(x, \theta^{(1)})$
Same for the other neurons.

$$y = f^{(3)}\left(f^{(1)}(x;\theta^{(1)}), f^{(2)}(x;\theta^{(2)});\theta^{(3)}\right)$$

Input for $f^{(3)}$ is the output of the functions of the previous level

↓

When we create a feedforward neural network we are building a model in which we are using the concept of function composition.

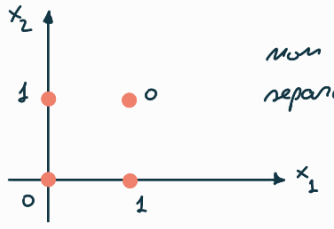


layers

only if all the activation functions are linear the whole network will be linear.

$f^{(1)}(x;\theta^{(1)})$ $\quad f^{(2)}(f^{(1)}(x;\theta^{(1)});\theta^{(2)})$ $\quad f^{(3)}(f^{(2)}(f^{(1)}(x;\theta^{(1)});\theta^{(2)});\theta^{(3)})$

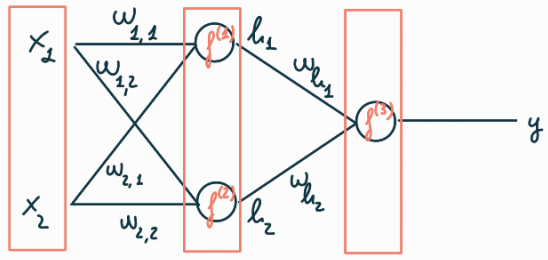
XOR problem



non-linearly separable dataset

A single perceptron will never learn the XOR function.
Surely we can use some non-linear kernel but we want to use neural networks.

Consider a neural network with two layers



input layer (2 nodes = dimension of the input)

hidden layer 1 (2 nodes)

output layer (1 node = dimension of the output)

activation function for the hidden layer : ReLU
activation function for the output layer : linear

$$ReLU(z) = \max(0, z)$$

usually activation functions are the same for a single layer and can be different between different layers.

So we have:

$$h_1 = ReLU(w_1^T x + w_{0,1}) = (w_{1,1} \; w_{1,2}) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + w_{1,0}$$
(w_1^T, x)

$$h_2 = ReLU(w_2^T x + w_{0,2})$$

We can rewrite this in matrix form

$$h = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = ReLU\left(\begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} w_{0,1} \\ w_{0,2} \end{pmatrix} \right)$$
(W^T, x, $+$, c)

$$y = (w_{h_1} \; w_{h_2}) \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + b$$
(w^T, h, b)

The whole model is:

$$y = w^T ReLU(W^T x + c) + b$$

$$= w^T ReLU\left(\begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} w_{0,1} \\ w_{0,2} \end{pmatrix} \right) + b$$

The parameters of the model are

$$\theta = \langle W, c, w, b \rangle$$

model non linear in θ

We have a parametric model. We now want to define an error function

$$\text{model: } y(x) = f(x;\theta) = w^T \max(0,\ W^T x + c) + b$$

We will use the Mean Squared Error (MSE) loss function:

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N} (t_n - y(x_n;\theta))^2$$

t_n: values in the dataset; $y(x_n;\theta)$: prediction of the network

Solution:

$$W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad c = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \qquad w = \begin{pmatrix} 1 \\ -2 \end{pmatrix} \qquad b = 0$$

Let's check the solution yield error = 0

$\bar{X}$: design matrix; set of all the possible values for the hidden layer for every input value

$$\bar{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}^T \qquad t = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

We are repeating the equation for each sample in the dataset in a compact way

$$\bar{h} = \text{ReLU}(\bar{X}W + \bar{c}) = \text{ReLU}\left(\begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{pmatrix}^C\right) = \text{ReLU}\left(\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{pmatrix}\right)$$

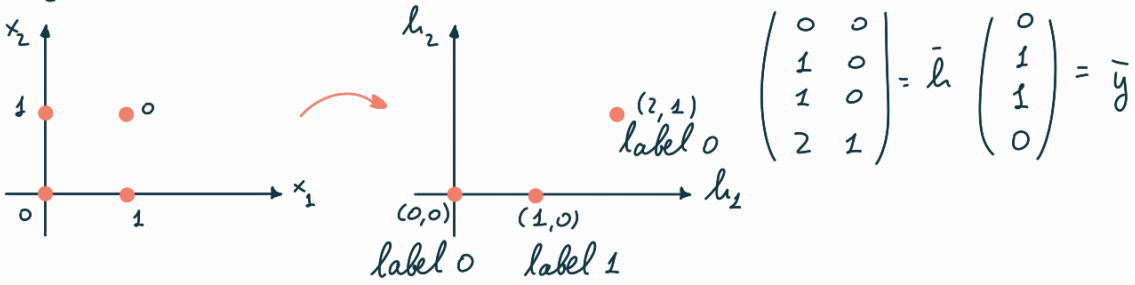
$$= \text{ReLU}\left(\begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}\right) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} = \bar{h}$$

$\bar{y}$: prediction of all the samples

$$\bar{y} = w^T \bar{h} + b = \begin{pmatrix} 1 & -2 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} + 0 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \bar{y}$$

the mean squared error comparing $\bar{y}$ with $t = (0,1,1,0)^T$ is exactly 0.

Let's plot the dataset $\{((0,0),0),((0,1),1),((1,0),1),((1,1),0)\}$ in the space defined by h_1 and h_2



$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} = \bar{h} \qquad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \bar{y}$$

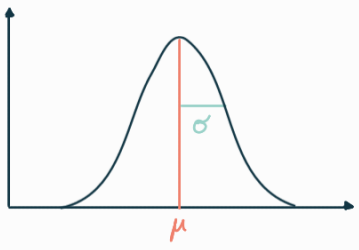
this transformation is good because the dataset is now linearly separable. This is the reason why a linear layer is sufficient: the dataset is linearly separable and a linear model (output layer) is more than enough.

Recall on some concepts

Gaussian function in general

$$f(x) = a\, e^{-\frac{(x-b)^2}{2c^2}}$$

for any real values a, b and non-zero c. The graph of a Gaussian is a bell curve. a is the height of the curve, b is the position of the center of the peak, c (standard deviation) controls the width of the bell.



Gaussian functions are often used to represent the probability density function of a normally distributed random variable with expected value $\mu = b$ (mean) and variance $\sigma^2 = c^2$. In this case the Gaussian has the form

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

Bernoulli Distribution

$$\left.\begin{array}{l} P(x=1) = \theta \\ P(x=0) = 1-\theta \end{array}\right\} \quad p(X=k,\theta) = \theta^k (1-\theta)^{1-k}$$

Before the choice of the Kernel was our responsibility (an hyperparameter); now the neural network found a way to learn a kernel by itself.

How to decide the layout of the network?
1. How many hidden layers it should have?
2. How many units each layer should have?
3. Which kind of units? (choice of the activation function)
4. Which kind of loss function?

1. How many hidden layers it should have?

The Universal approximation theorem states that: a FNN with a linear output layer and at least one hidden layer with any "squashing" activation function (e.g. sigmoid) can approximate any Borel measurable function with any desired amount of error, provided that enough hidden units are used.
It also works for other activation functions (e.g. ReLU).

Borel function: a map $f: X \longrightarrow Y$ between two topological spaces.

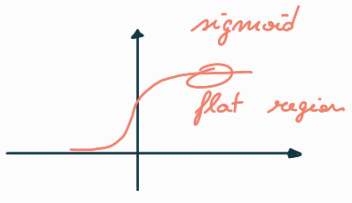
2. This theorem does not say how many neurons we should have in the hidden layer.

We know that a Boolean function of n variables can be emulated by a network with just one hidden layer of 2^n units (order of 2^n units).
For other functions we can't tell.

Unfortunately short networks are difficult to train. In practice, a network with many layers is easier to train and produces better results.

3. Which kind of units?
4. Which kind of loss function?

From one side, the universal approximation theorem tells us to use non-linear "squishing" functions, from the other side using any function that limits the input in a range is dangerous because of the flat areas. When you compute the gradient of a flat region you will have saturation: saturation means the gradient is close to zero.



sigmoid limits the input from $(-\infty, +\infty)$ to $(0,1)$

Let's see how to properly design a neural network. We need to choose a loss function (4), that will be optimized following the maximum likelihood principle, and the units for each layer of the network (3)

The choice of network output units and cost function are related

Which kind of units?

Focus on the output layer

Let $h = f(x; \theta^{(n-1)})$ be the output of the hidden layers (output of the last hidden layer)

- what is the function that returns the output given h and the parameters of the last layer $\theta^{(n)}$ $y = f^{(n)}(h; \theta^{(n)})$?
- which cost function $J(\theta)$?

Artificial Neural Networks can be used to solve any kind of machine learning problem.

- Regression
- Binary classification
- Multi-class classification

Depending on the task we are solving we will need a different kind of unit in the output layer and this will lead to the adoption of a particular error function:
Depending on the task we are solving there is an optimal (suggested choice of the combination of the output model with the cost function.

③ choice of output units
- Regression → ④ which kind of error function?
- Binary classification → ④ which kind of error function?
- Multiclass classification → ④ which kind of error function?

Regression

③ Wich kind of output units ?

For regression problems it is convenient to consider a linear model as last layer for the neural networ in which the function is an *identity*

$$y = W^T h + b$$

h → output of the previous hidden layer

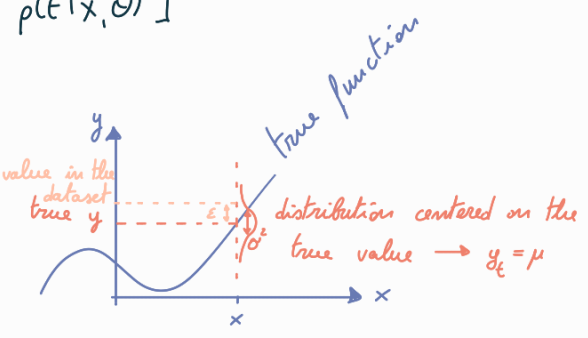
We suppose that other layers will introduce non-linearities

④ Which kind of error function ?

The model implicitly defines a conditional distribution $p(t|x,\theta)$
The cost function will be optimized following the maximum likelihood principle

$$J(\theta) = \mathbb{E}_{x,t\sim\hat{p}_{data}} [-\ln p(t|x,\theta)]$$

In a regression problem :



Target values t in the dataset are given by the true function $y(x;w)$ affected by additive noise ϵ.

$$t = y(x;w) + \epsilon$$

We assume that the noise follows a gaussian distribution.
We just described the same problem from a probabilistic point of view.

$$P(t|x,w,\beta) = \mathcal{N}(\epsilon | y(x;w), \beta^{-1})$$

variance is described in terms of precision β ; precision is the inverse of the variance

$P(t|x,w,\beta)$ → likelihood

$y(x;w)$ → the distribution is centered on the true value (zero-mean)

Now we can try to maximize the likelihood (minimize the negative log likelihood)

$$\text{maximize } P(\{t_1,\dots,t_N\}|x_1,\dots,x_N,\theta,\beta) = \prod_{n=1}^{N} \mathcal{N}(t_n | f(x_n;\theta), \beta^{-1})$$

under the hypothesis that the samples are independent and identically distributed we can use the total probability theorem

or equivalently :

$$\text{minimize } -\ln P(\{t_1,\dots,t_N\}|x_1,\dots,x_N,\theta,\beta) = \sum_{n=1}^{N} -\ln \mathcal{N}(t_n | f(x_n;\theta), \beta^{-1})$$

$$= \underbrace{-\beta \frac{1}{2} \sum_{n=1}^{N} (t_n - f(x_n;\theta))^2}_{E_D(w)} \underbrace{- \frac{N}{2} \ln(2\pi\beta^{-1})}_{\text{do not depend on } \theta}$$

transformation of the whole network

$E_D(w)$ is what we need to minimize
Maximum likelihood (minimum log likelihood) under zero-mean Gaussian assumption correspond to *least square error minimization*

So, for a regression problem under zero-mean gaussian assumption the error function naturally becomes the least square error

Using linear units and the mean squared error as error function is a good design choice: this usually does not yield the gradient to saturate. The gradient saturates ($\to 0$) when there are flat regions and this does not happen here because the linear function does not have flat regions.

maximum_likelihood (gaussian_zero_mean) = minimize (mean_squared_error)

Binary classification

We assume, as before, to encode the two classes with 0 and 1. Under this assumption we can notice that we have the exact same situation as with the Bernoulli distribution in probabilistic classification.

③ Which kind of output units ?

A Bernoulli process could only have two outcome, 0 and 1. An event will have a probability of outcome 0 that is $\theta \in [0,1]$ and a probability of outcome 1 that is $1-\theta$. We can use a sigmoid function $y = \sigma(W^T h + b)$ to model this.

$$\theta \text{ is given by } \sigma(\alpha) \quad \text{with } \alpha = W^T h + b$$

④ Which kind of error function ?

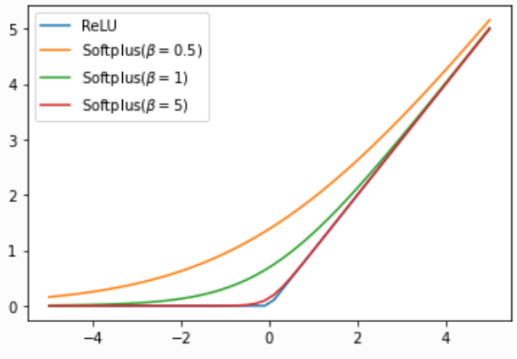
The likelihood we need to maximize corresponds to a Bernoulli distribution.

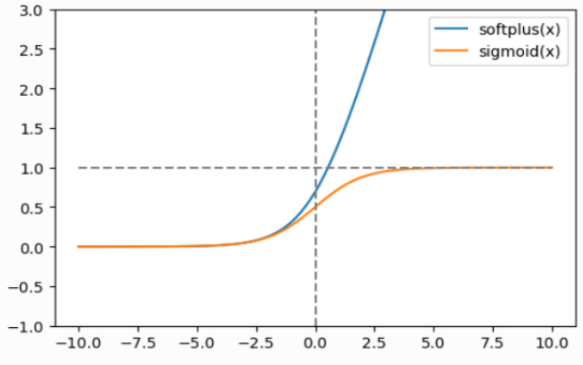
$$J(\theta) = \mathbb{E}_{x,t \sim D} \left(- \ln p(t|x) \right) \longrightarrow \text{log likelihood}$$

$$p(t|x) = p(X = k, \theta) = \theta^k (1-\theta)^{1-k} \longrightarrow \text{Bernoulli distribution}$$

$$\begin{aligned} - \ln p(t|x) &= \theta^k (1-\theta)^{1-k} \\ &= - \ln \sigma(\alpha)^t (1-\sigma(\alpha))^{1-t} \\ &= - \ln \sigma((2t-1)\alpha) \\ &= \text{softplus}((1-2t)\alpha) \end{aligned}$$

We come to the definition of a function called softplus





What can we say about this combination ? We have saturation (because of the sigmoid) but this only happens when we are close to the solution

maximum_likelihood (bernoulli) = minimize (softmax)

Multiclass classification

We assume the n classes are mapped with $i \in [0, n-1]$. We have a multinomial distribution on the output: we should have an output layer with one unit for each class, each outputting the probability of the corresponding class being the result; the sum of all the probabilities should therefore be 1; one of them should be close to 1 and the others very low if we have no uncertainty.

③ Wich kind of output units ?

We can use the softmax activation function

$$y_i = \text{softmax}(\alpha_i) = \frac{e^{\alpha_i}}{\sum_j e^{\alpha_j}}$$

⑤ Which kind of error function ?

$$J_i(\theta) = \mathbb{E}_{x,t \sim D} \left[-\ln \text{softmax}(\alpha_i) \right] \quad \text{with} \quad \alpha_i = w_i^T h + b_i$$

Even in this case, the gradient saturates when we have minimal error (we are close to the solution).

Summary for the output units

Task	Output units	Loss function
Regression	Linear	Mean Squared Error
Binary classification	Sigmoid	Binary cross-entropy
Multiclass classification	Softmax	Categorical cross-entropy

If you don't have any other information about the problem these are the guidelines

What about the hidden units?

This is another aspect that we should better understand but we don't have theoretical principles yet, only intuitions. In general we don't know what is the best choice for the activation functions in the hidden units. The only guideline is to use the Rectified Linear Unit (ReLU) if you have no other information about the problem.

$$g(\alpha) = \max(0, \alpha)$$

- easy to optimize
- not differentiable in 0 but this does not cause problems in practice

Sigmoid and hyperbolic tangent considerations

Sigmoid: $g(\alpha) = \sigma(\alpha)$
Hyperbolic tangent: $g(\alpha) = \tanh(\alpha)$

These two functions are closely related as $\tanh(\alpha) = 2\sigma(2\alpha) - 1$
These two functions have flat regions, so with certain output units they can saturate. If the output units are linear, it is no more guaranteed that the saturation will happen only when we are close to the solution.
When using these two functions we have to consider the last layer and the error function

- units saturates easily
- gradient based learning is very slow
- hyperbolic tangent gives larger gradients with respect to the sigmoid
- useful in recurrent networks, autoencoders,...